採択課題番号：IDEAS201903

普及型フィールドセンサによるOGC標準データの災害情報サービスへの提供の研究

保科紳一郎*、金帝演*、神田和也*，本多潔**，竹島喜芳**，伊勢田良一***，
鶴岡工業高等専門学校，中部大学国際GISセンター，***NTT ドコモ

1. はじめに

近年の地球環境の変動が、短時間で局所的な降雨の集中、いわゆる「ゲリラ豪雨」を頻発させている。都市においてはアンダーパスの冠水による人的な被害や、農村部においては土砂災害などの被害を及ぼしている。このような災害予測を行うためには数多くの観測点を設ける必要があり、行政などで整備が進められている。その様ななかで観測点を増やすためには、データ精度を許容範囲で保証し安価で低消費電力、かつ、耐環境性を備えたフィールドセンサを数多く設置することが期待されている。

現在のフィールドセンサの多くはサービスと一体・不可分なものとして提供されている。他社のフィールドセンサを他のサービスに流用することは困難であり、その結果フィールドセンサの普及は限定され、取得したデータも抱え込まれてしまい、他の組織の提供するサービスやデータと融通しあうことは容易でない。

各フィールドセンサが実装するデータ送信方式を、一般に公開されている標準化形式に従ったデータ書式、データ送受信APIに準拠させることで、異なる装置・サービスであっても容易にデータの相互参照が可能となり、現在不可分の関係となっているフィールドセンサとサービスのつながりをより柔軟なものに変えることができると考えられる。その結果、フィールドセンサおよびそれを使ったサービスの低価格化が促進され、フィールドセンサの設置数が増えると期待される。現在進められている標準化の事例として、農業分野では総務省が中心となり環境情報の表現方法について標準化が進められている。この標準化にはOGC(Open Geospatial Consortium) [1]標準のSOS(Sensor Observation Service) [2]規格で定めたデータ送受信API(以後SOS APIと呼ぶ)を採用している。

本研究の目的は、フィールドセンサの中核となるマイコン（例 Arduino UNOやBeagleBone Black等)にSOS API [3]を容易に実装することができるソフトウェアパッケージを提供することにより、標準化されたインタフェースを実装するフィールドセンサの開発を加速することである。本研究の中で、フィールドセンサで利用されている主要な通信形態を取り上げて、それらに対応するSOS APIの導入方法について検討した。また、本研究の成果を本とした既存のフィールドセンサシステムのSOS APIの導入例として、「KOSEN版ウェザーステーション」 [4]へのSOS APIの導入を示した。また本研究により得られたソフトウェアパッケージの公開を行なった。

2. 標準化とその目的

2-1 OGC標準SOS API

OGC標準において、SOSは情報を保存するデータベースにアクセスするAPIを提供する。図1中に実線赤で示されているのは、「SOSサーバ」であり、SOSサーバはSOSインタフェースとDB(データベース)で構成されている。OGC標準においてSOSサーバはデータの収集と配信を担う。SOSサーバ(SOS API)のVersionによっ

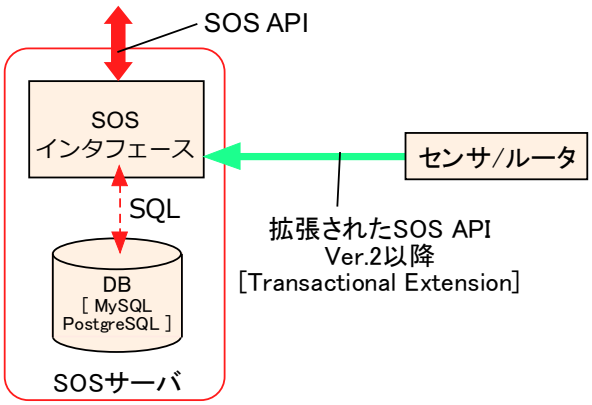

図 1 SOSサーバの構成

		記述形式	
		XML形式	JSON形式
通信手順	通常	×	△
	テンプレート	△	◎

通信データの相対的な評価：×多い、△少ない、◎最も少ない

表 1 SOS APIの通信手順と記述形式の評価

て SOS API が対応する機能に違いがある。Version 1 では DB からのデータ参照のみであり、データを DB へ入れることは想定されていない。DB へデータを入れることが可能となるのは、機能が拡張された Version 2 以降となる。Version 1 では SOS インタフェースを介さずに直接 DB にデータを入れる手法を各自で実装する必要があった。

SOS API の記述形式として、XML 形式と JSON 形式の二つが定められている。通信手順として SOS API では XML 形式のデータを送信するために、HTTP 通信 POST メソッドで SOAP プロトコルを利用している。また JSON 形式のデータを送る際にも POST メソッドを利用している。ここではこれらの通信手順を SOS API での「**通常**」の通信手順と呼ぶことにする。通常の手順に対して「**テンプレート**」を利用した場合を比較する。テンプレートとは、データ送信を開始する前にデータ転送に伴うデータ並び等のデータ記述法などのメタ情報をまとめたデータである。テンプレートを用いることで、データ送信時に重複するデータを送る必要が無いことから、送信データ量を低減する効果がある。表 1 にデータ送信時の通信データ量を、通信手順とデータ記述形式毎に評価した表を示す。通信量の低減が最も期待できるのは、通信手順として「**テンプレート**」を用いて、記述形式として「**JSON 形式**」を採用した場合である。

図 2､3 では､XML 形式で SOAP プロトコルを用いてデータを送信する｢通常｣の SOS API｢InsertObservation｣と、JSON 形式でテンプレートを使ってデータを送信する SOS API「InsertResult」の通信手順を示す。図 2、3 において緑線で囲まれている部分は SOS サーバの情報取得とセンサを SOS サーバに登録する API「InsertSeneor」の発行である。赤線で囲まれている部分は測定結果を SOS サーバに送信する部分である。緑のセンサ登録は一度行えばよく、連続的にデータを送るには赤の手順を繰り返すことになる。図 3 のテンプレートを用いる方法は青で囲まれている手順でテンプレートの設定を行う。図 2 に比べるとテンプレートの設定の手続きが多くなるが、テンプレートの設定は一度行えばよいので、全体的な通信量の増加にはつながらない。通信量は、測定結果を繰り返して送信する赤の領域のデータ送信量に大きく影響される。赤のデータ送信部の部分で、テンプレートを使ってデータ送信時に重複するデータを省略し、JSON 形式でデータ記述することは、通信量の低減に大きな効果を与える。

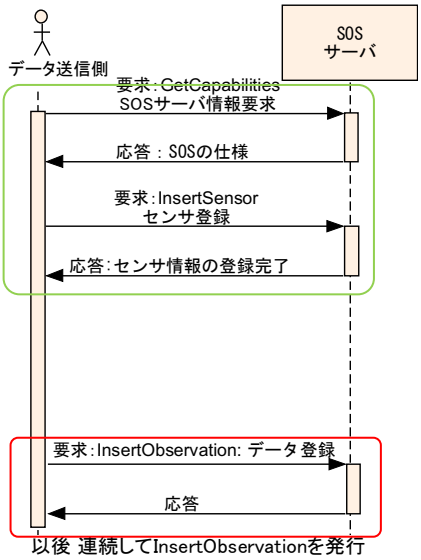

図 2　InsertObservation(XML 形式)

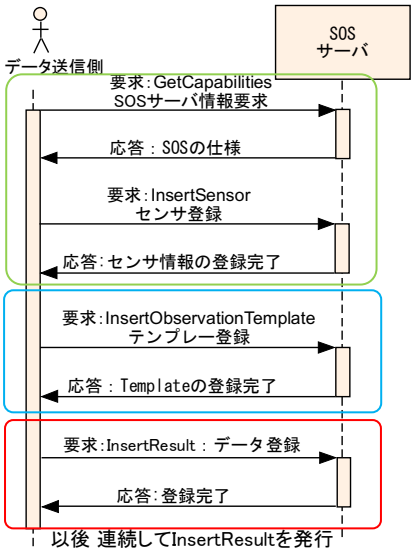

図 3　InsertResult(JSON 形式)

2-2　OGC 標準及び SOS API の普及例

現在､防災科学研究所が中心となって防災関連の情報収集､評価そして警報などの情報発信の試みがなされている。地域の住民・自治体などと連携して防災関連の情報共有を目的に「災害リスク情報プラットフォームの研究開発」を行なっており、e コミュニティ・プラットフォーム(図 4) [5]を立ち上げ運用している。e コミュニティ・プラットフォームのシステムでは、防災関連情報をより効果的に用いるために、ある位置における情報と地図情報を統合して運用する地理情報システム(GIS：Geographic Information System)が取り入れられており、自治体や町内会のレベルで防災情報の可視化ができるような仕組みが構築されている。本システムの中心となっている g サーバでは、データの送信は OGC 標準 SOS API [6]が採用されている。気象

情報等を SOS サーバに蓄積することで g サーバとの連携を取ることもできる。

データの記述方法として、平成 26 年度末に、国の取り組みとして高度情報通信ネットワーク社会推進戦略本部（IT 総合戦略本部）・新戦略推進専門調査会農業分科会(図 5)［7]において、農業情報創成・流通促進戦略が示された。その際に提案された農業情報の記述方法に OGC 標準が採用された。具体的なデータの記述方法は、図 6 の「個別ガイドライン」としてまとめられている。例えば、図 7 に示すように個別ガイドラインでは、SOS API で用いるデータの記述に必要なデータの属性を表現する URI、や observationProperty の名称、単位が定義されており、本研究でデータ(気温、水位等)の記述を行う際は、個別ガイドラインを採用している。個別ガイドラインには SOS API の記述例も同時に表示しており、SOS API でのデータの記述の参考とした。

図 4　e コミュニティ・プラットフォーム

図 5　IT 戦略本部

- ＜GL4＞農業情報のデータ交換のインタフェースに関する個別ガイドライン（第2版）
（平成29年3月10日新戦略推進専門調査会 データ活用基盤・課題解決分科会取りまとめ）　（Word形式 ／ PDF形式）

別紙１　Capabilities.xml　（XML形式）
別紙２　SensorML.xml　（XML形式）
別紙３　Observations_and_Measurements.xml　（XML形式）
別紙４　GetResultTemplateResponse.xml　（XML形式）
別紙５　GetResultResponse.xml　（XML形式）
別紙６　GetCapabilities.xml　（XML形式）
別紙７　DescribeSensor.xml　（XML形式）
別紙８　GetObservation.xml　（XML形式）
別紙９　GetResultTemplate.xml　（XML形式）
別紙１０　GetResult.xml　（XML形式）
別紙１１　環境情報のデータ項目URIリスト.html　（HTML形式）
別紙１２　メタ情報のパラメータURIリスト.html　（HTML形式）

図 6　個別ガイドライン

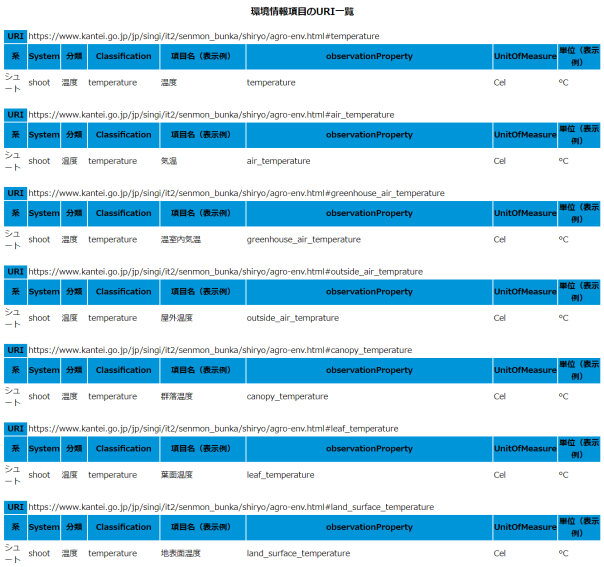

図 7　個別ガイドライン データ記述方法

3.　SOS 対応センサの開発

3-1 通信形態と開発環境

本研究において SOS API に係わる通信形態として検討した通信形態を図 8 に示す。図 8 はフィールドセンサのデータ送受信で採用される主要な通信形態を示している。検討する通信形態中ではデバイスとして、データの取得と送信を行う「フィールドセンサ」、データ転送の中継やプロトコル変換を行う「ルータ」を想定する。図 8 中の斜線部に SOS API の実装を行う。

SOS API の開発・実証環境として図 9 を示す。開発環境を構築する上で一番の課題は、機能の検証を行うための SOS サーバの構築である。SOS サーバ自体は 52°North からフリーソフトとして提供されており [8]、インストール方法・インストール後の設定も指示されている。しかし、ソースファイルから SOS サーバを構築することは非常に難しい。本研究では開発環境を容易に構築するために、OSGeo 財団のプロジェクトで提供されている Linux(Ubuntu16)の LiveDVD である OSGeoLive を利用している。OSGeoLive [9]では地理空間情報に係るフリーソフトをインストールした状態で提供しており、OSGeoLive に 52°North の提供する SOS サーバ環境も含まれている。Windows10 上に VMWare を使って仮想マシンを作り、そこに OSGeoLive をインストールして使用している。自由に設定できる SOS サーバを PC 上に構築できれば、SOS API の動作検証も容易となり、プログラム開発がより効率的となる。また、52°North SOS サーバをインストールすると一緒にインストールされるテストクライアント(図 10)を使って API の動作検証を行うこともできる。テストクライアントは web ブラウザより操作を行うことができ、例えば図 10 は SOS API の一つである「GetCapabilities」を発行する様子である。テストクライアントを使って SOS API の動作を確認できることは SOS API の開発において、その動作を理解するのに非常に有効であり、特に SOS API を始めて利用する開発者には是非とも利用してほしい。

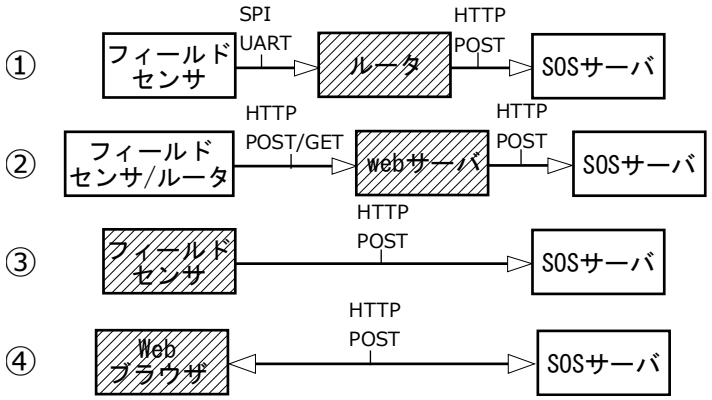

図 8　フィールドセンサの通信形態

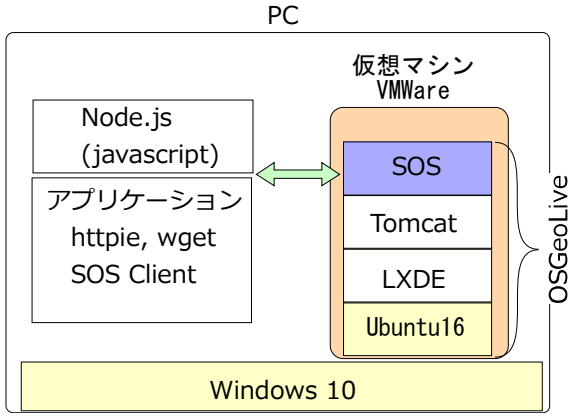

図 9　開発環境の構成

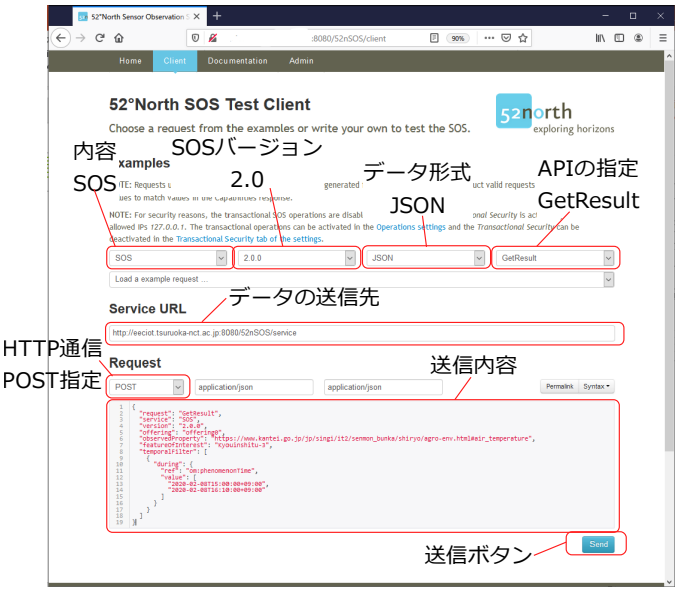

図 10　52°North SOS サーバの
テストクライアント

3-2 SOS API 実装の要点

データ送信毎に使用される SOS API は定型的であることから、SOS API のひな形を用意してデータ送信毎に必要な部分(例えば、時刻や測定値等)を書き換えることにすると、簡単な文字列変換プログラムにより SOS API を作成することができる。そこで、SOS API の「ひな形」と、書き換えるデータを書き出したリスト(以後「テーブル」と呼ぶ)を基にして、「ひな形」を「テーブル」に従って書き換えるプログラムを作成した。図 11 にひな形とテーブルの例を示す。図 11(a)はセンサを SOS サーバに登録する API である「InsertSensor」を元にしたひな形である。InsertSensor にはフィールドセンサに係る各種情報が記述される。例えば図 11(a)中のフィールドセンサ名を示す「offering」という項目がある。そこで SOS API 中の offering を記入する個所を「%offering%」とする変数で表している。そして図 11(b)で示されるテーブルに従ってひな形の必要個所を書き換える。テーブルは JSON 形式で記述されている。図 11(a)のひな形中の変数%offering% は、図 11(b)テーブル中の変数%offering%の値 offering0 に置き換えられる。同様にしてテーブルを基にしてひな形内の変数%id%は ws00、%procedure%は ws00 に書き換えられる。これらの書き換え機能をブロック図で図 12 のように表す。図 12 を JavaScript 言語で記述した例を図 13 に示す。クラス ReplaceStrings_text ではインスタンス時に、SOS API のひな形を記述したテキストデータ変数「text」を渡してコンストラクタを起動する。ReplaceString_text のメソッド replace_table に変換用テーブルを表した JSON 形式データ変数「table」を渡すと、戻り値として変換用テーブルに従って文字列置換を行なった SOS API のテキスト文字列を返す。

```
<?xml version="1.0" encoding="UTF-8"?>
<env:Envelope>        SOS API名
  <env:Body>
    <swes:InsertSensor service="SOS" version="2.0.0“

      <swe:label>offeringID</swe:label>
      <swe:value>%offering%</swe:value>

    </swes:InsertSensor>
  </env:Body>
</env:Envelope>
```

(a)　SOS API のひな形(XML/JSON 形式)
InsertSenor(センサの登録)

```
{
   "id":"ws00",
   "procedure":"ws00",
   "offering":"offering0",
   "feartureOfInterest":"Kyouinshitu-3",
   "x":"139.79777552451446",
   "y":"38.709857896925634",
   "z":"15",
   "temp":"ws00/template/temp",
   "humi":"ws00/template/humi"
}
```

(b)　変換用テーブル(JSON 形式)
センサの基本情報

図 11　SOS API 作成のための
「ひな形」と「テーブル」

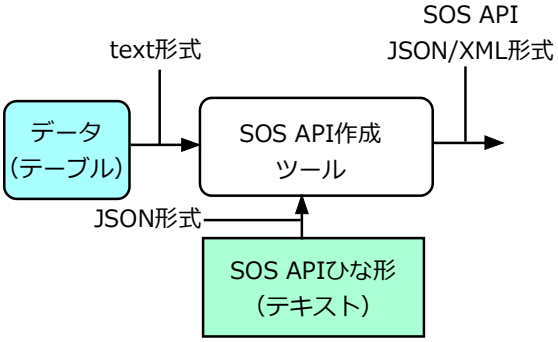

図 12　SOS API の作成ブロック

```
/*
    引数text SOS APIのひな形
*/
function ReplaceStrings_text(text){
   this.text = text;   //SOS APIのひな形（TEXT形式）を保持する。
};

/*
    引数table SOS APIを書き換えるテーブル(JSON形式)
*/
ReplaceStrings_text.prototype.replace_table = function(table){
   let txt = this.text;
   for(i in table){
      let string=`%${i}%`;
      regexp = new RegExp(string,'g');
      txt = txt.replace(regexp,table[i]);   //文字列置換
   }
   return txt;
}
```

図 13　文字列置換プログラム

3-3 通信形態毎の SOS API 実装

(1) 通信形態①での SOS API の実装

図 14 で示される通信形態①について SOS API の実装では SOS API をルータに実装する。ここでルータとしては、Linux 等の OS が搭載されており、Node.js がインストールされているとする。具体的には Raspberry PI や BeagleBoneBlack 等の Linux 搭載マイコンを想定している。SOS API の実装は Node.js 上で動く JavaScript 言語のプログラムとして実現する。図 14 にルータへの SOS API の実装したブロック図を示す。フィールドセンサより SPI や UART でルータに測定結果が送信される。ルータは送信データを受信し、送信データを SOS API の書式に従って書き換える。図 14 の赤点線部は 3-1 節で説明した SOS API の作成ブロックであり、図 12 に従って作成したプログラムを流用することができる。

(2) 通信形態②での SOS API の実装

ネットワーク管理やセキュリティの関係上 SOS サーバを直接外部に公開することは難しいことが多い。そこで、フィールドセンサから送信されるデータを外部に公開されている web サーバで受け、web サーバはリバースプロキシ機能を使って受信したデータを指定された URL にデータを転送する。図 15(a)の web サーバのデータ転送の様子を図 15(b)に示す。図 15(b)に示すように web サーバのデータ転送先は SOS 代理サーバとする。SOS 代理サーバの構成を図 15(c)に示す。SOS 代理サーバは web サーバを経由して受信したデータをもとに SOS API を構成し SOS サーバにデータを登録する。図 15(c)の HTTP サーバは Node.js の Express モジュール [10]を使って作成している。HTTP サーバ以降の構成は、通信形態①を実現する図 14 のブロック図とほぼ同じであることが分かる。そこで SOS 代理サーバを通信形態①で用いたプログラムを流用して作成した。

(3) 通信形態③での SOS API の実装

通信形態③はフィールドセンサが直接 SOS サーバにデータを送信する。そこでフィールドセンサに SOS API を実装する。フィールドセンサのコントローラとして、省電力化が求められて Raspberry Pi のような高機能なマイコンではなく、Espressif Systems 社が製造する ESP8266 [11]のようなマイクロコントローラと TCP/IP スタック、Wi-Fi ネットワーク機能を有する小型で通信機能を有するマイコンを想定する。このようなマイクロコントローラに SOS API を実装するために、本研究では、マイコンに依存しない図 12 で示されるような SOS API 作成ブロックを C/C++言語で作成した。

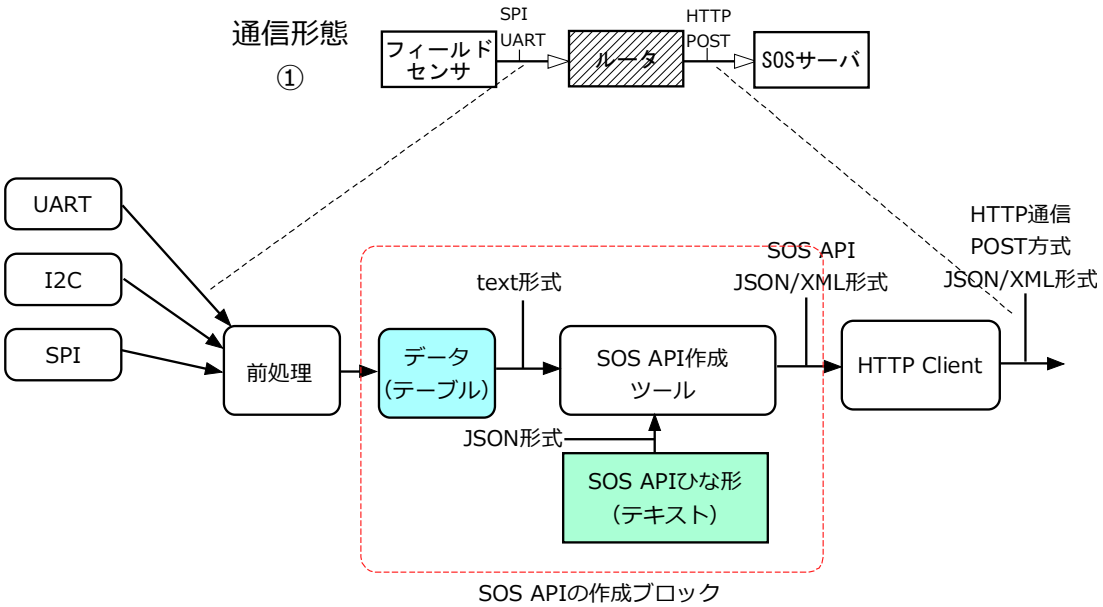

図 14　SOS API の実装(通信形態①)

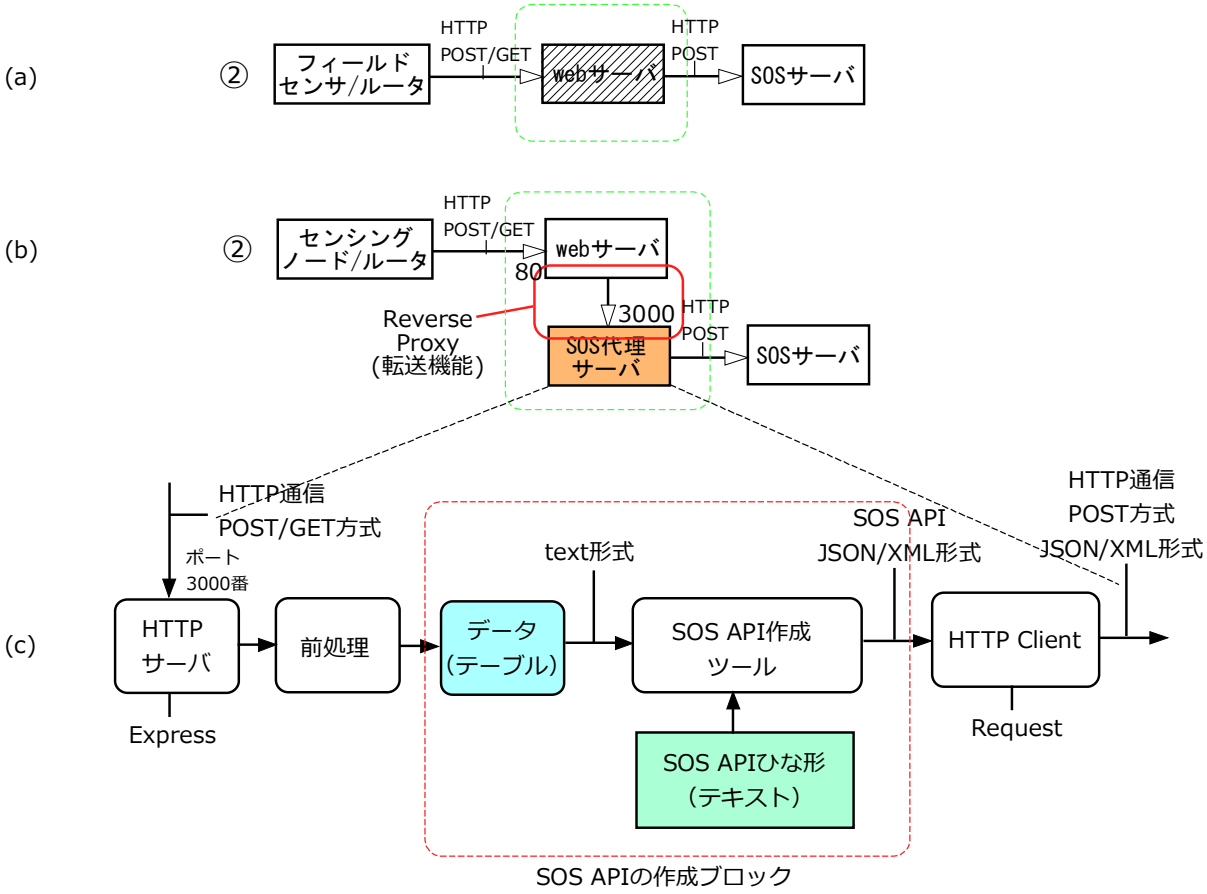

図 15　SOS API の実装(通信形態②)

```
//
// InsertResult APIの例
//                                    Rプレッフィスク
String insertresult = R"({
  "request": "InsertResult",
  "service": "SOS",
  "version": "2.0.0",
  "templateIdentifier": "tsuruoka/arduino/00/temp/",
  "resultValues": "%DATA%"
})";
```

図 16　C/C++でのテンプレートの例
(SOS API InsertResult)

```
/* SOS APIクラス */
class SOS_API {
private:

protected:
    String original;//APIの編集前
    String copy;//編集用
public:
    //APIのテンプレートの読み込み
①--     SOS_API(String template);
    //%DATA%をwordに書き換える
②--     String write(String word);
    ...
};
```

図 17　SOS API 作成クラスの作成例

図 16 は SOS API(InsertResult)のひな形を示す。R プレフィックスを用いた文字リテラルを使用することで、C/C++のソースプログラム中に直接 JSON 形式のデータを記述することができる。変数%DATA%の部分が実際のデータに置き換えられる。図 17 のように C/C++で SOS API を実装するクラス SOS_API を作成した。図 17 中のクラス SOS_API の①に示されるコンストラクタに引数として SOS API のひな形を渡し SOS API オブジェクトを作成する。図 17 中の②に示される SOS API オブジェクトの write メソッドは、引数として JSON 形式の文字列変換用テーブルを持つ文字型変数を渡し、図 12 の文字列置換を行なって作成した SOS API の文字

列を戻り値として返す。

(4) 通信形態④での SOS API の実装

先に上げた通信形態①～③と異なり、通信形態④では web ブラウザから SOS サーバにアクセスしてデータを参照する。そこで、web ブラウザで参照した HTML ファイルから SOS API を実装する JavaScript プログラムを実行する。図 18 の赤点線で示される部分は、図 12 で示される SOS API の生成ブロックである。通信形態は、サーバサイドとクライアントサイドの違いこそあるが、通信形態①とほぼ同じである。そこで通信形態①で使用している JavaScript のプログラムをそのまま流用することができた。web ブラウザと SOS サーバとの通信は jQuery ライブラリ [12]で提供されている Ajax 通信モジュールを使用している。

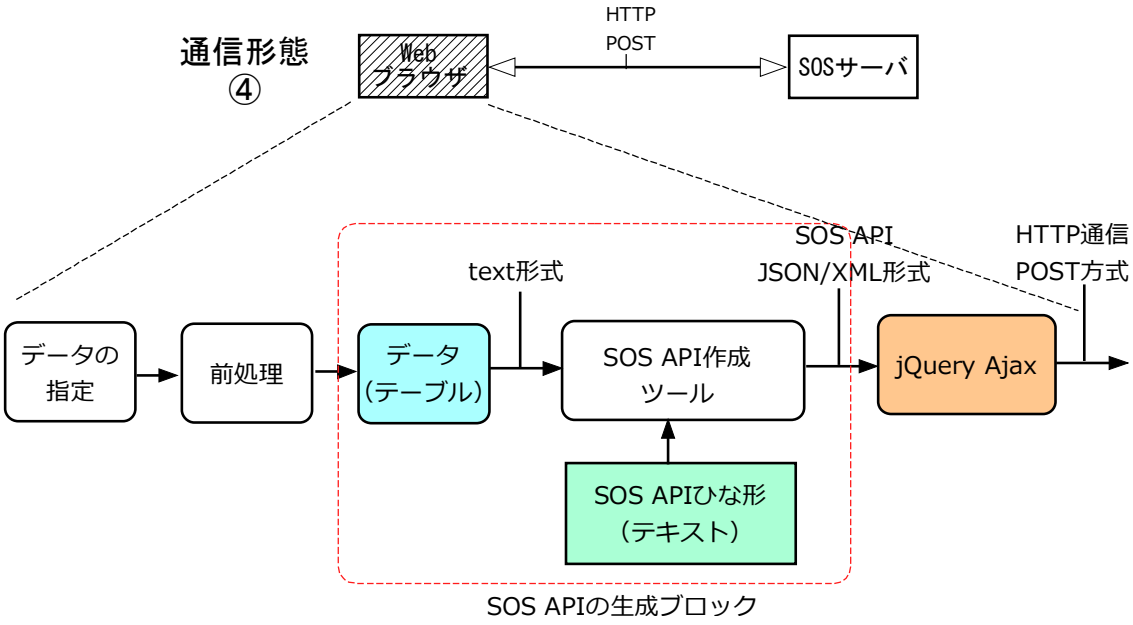

図 18　　SOS API の実装(通信形態④)

4.　SOS API 実装例(「KOSEN 版ウェザーステーション」への SOS API の実装)

4-1　「KOSEN 版ウェザーステーション」の概要と通信形態

平成 26 年度に鶴岡高専において農業用の簡易気象観測装置の試作を行い、これを元に他の高専(仙台、阿南、香川高専等)・高専機構本部の支援を受けて改良発展させて現在の「KOSEN 版ウェザーステーション」となった。現在はキット化されて販売がされており、農業分野だけでなく防災分野への展開、工学分野の IoT 教育への導入が進められている。

図 19 は、現在鶴岡高専で開発・改良が進められている KOSEN 版ウェザーステーションである。風向・風速計、雨量計等のセンサ群、発電用の太陽光パネル、コントロールボックス内には電源及びセンサ群の制御装置を収容している。現在、その仕様上ウェザーステーションで取得したデータは LTE 回線で SORACOM Beam(http://beam.soracom.io:8888/)へ HTTP 方式 POST メソッドでデータを送信している。データの流れは図 20(a)のようになっている。

4-2　「KOSEN 版ウェザーステーション」の SOS API 対応

4-1 でも述べた通りウェザーステーションの仕様の上で SORACOM Beam へのデータ送信は定められており、ウェザーステーションに組み込まれているプログラムの変更無しに転送先を SOS サーバに変更することができない。そこで、図 20(b)に示すように SORACOM Beam からの転送先を指定の Web サーバ(鶴岡高専 Web サーバ、http://xxxx.tsuruoka-nct.ac.jp/)に変更し、HTTP 方式 POST メソッドで送信する。図 20(b)の通信形態は 3-3 (3)で示したものと同様であるので、3-3 (3)で作成したプログラムを流用する。高専 Web サーバで受信したデータを SOS 代理サーバで SOS API に変換し SOS サーバに登録する。

図 21(a)に SOS 代理サーバが、SORACOM Beam、高専 Web サーバを経由して受信したデータの一例を示す。データはすべてテキストデータで表現される。データ形式として JavaScript のオブジェクト型データ中に測定

回数分大きさを持つ配列(配列名 items)を持ち、配列の内容は測定結果を入れたオブジェクト型データである。KOSEN 版ウェザーステーションは通信に要する電力消費を抑えるために、一度のデータ送信で二又は三回分の測定結果をまとめて送信する。図 21(a)に示すように三回分の測定結果(測定結果 1～3)が収納されている。図 21 (b)に三回分の測定結果をまとめた SOS API(InsertResult)と SOS サーバからの返答を示す。SOS サーバからの応答からデータの登録が正しくできていることが確認できる。以上から、三章の通信形態毎の検討結果を利用することにより、様々な通信形態に対応した SOS API の実装がより容易になったことを示した。

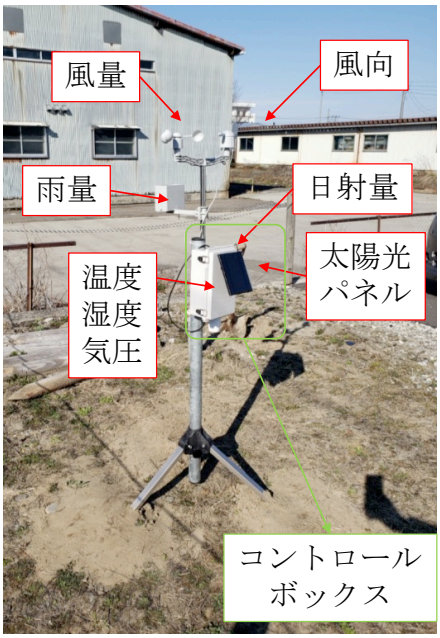

図 19　KOSEN 版ウェザーステーション
（フィールドセンサ部）

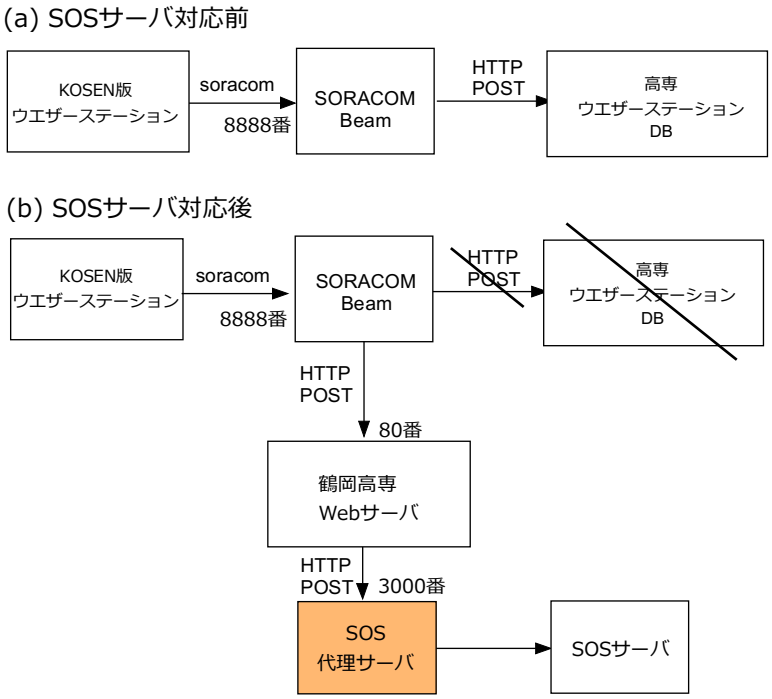

図 20　KOSEN 版ウェザーステーションの通信

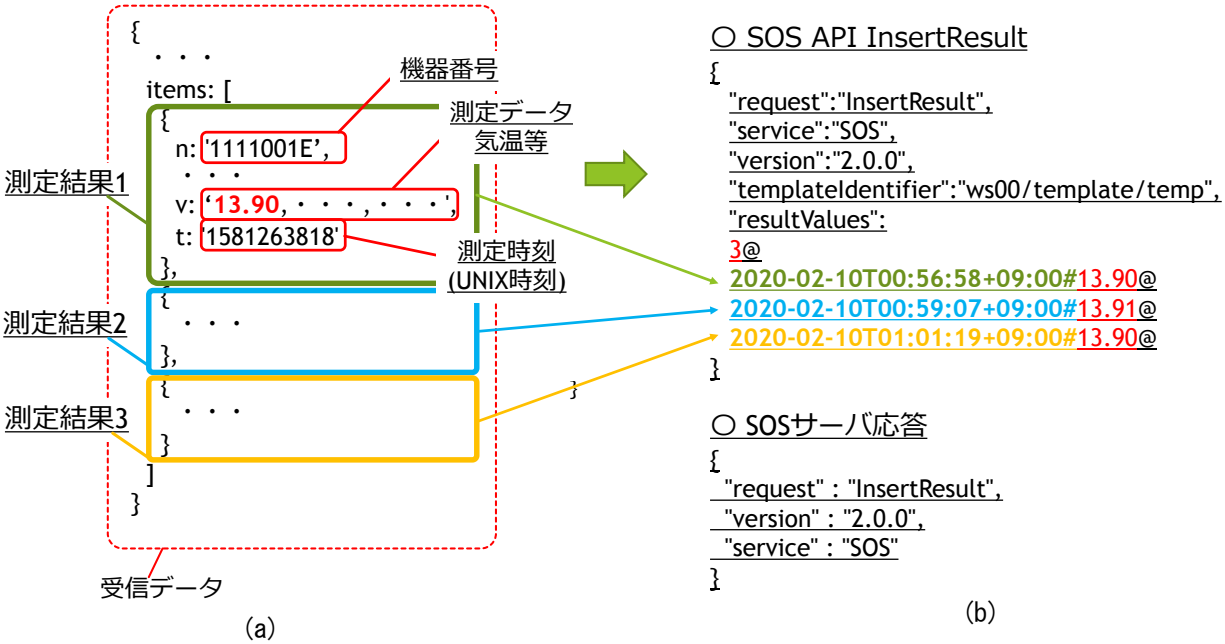

図 21　KOSEN 版ウェザーステーション
のデータと SOS API(InsertResult)

5. まとめ

近年の気候災害の激甚化に対応すべく災害予測の精度向上のため、より細かい地点における防災情報の収集が必要であり、そのために観測装置の低価格化と低価格化を促進するための通信手法やデータ形式の標準化の推進が必要である。本研究では通信手法とデータ形式の標準化を促進するために、地理情報の通信手法として広く採用されている OGC 標準に準拠したデータ送受信手法である SOS API、そしてデータ形式の標準化として IT 戦略本部で採用されている農業情報の標準化を採用している。これらの通信手法とデータ形式の標準化の具体的なフィールドセンサへの実装方法を示し、標準化の普及によりセンサの低価格化を目指すものである。

本研究では、SOS API の実装手法を開発環境の構築までを含めて表すことができた。開発環境の構築は SOS API の実装に大きな課題である。しかし、開発環境として OSGeoLive DVD を用いることで OGC 環境を容易に構築できることを示した。SOS API の実装プログラムの開発には PC 内に VMWare 等を使って仮想マシンを構築して、仮想マシン上に OSGeoLive をインストールすることで、SOS サーバ、web サーバ等を同一マシン上に構築するこができる。この環境を利用することで SOS API に係わるプログラムの開発を効率的に進めることができる。

フィールドセンサに係わるデータ送信の通信形態の代表的な例を挙げ、それらの代表的な通信形態毎に SOS API の実装の要点を示した。SOS API の内容は XML 形式又は JSON 形式のテキストデータであること、フィールドセンサから送られる観測データは定型的であること、これらから SOS API を記述した XML 形式や JSON 形式の文字列の一部を、観測データに従って随時書き換えて SOS API を作成する手法を提案した。書き換えるプログラムは JavaScript で記述されておりサーバ側では Node.js、クライアント側では web ブラウザ上で実行可能なモジュールとなっており、本研究で想定した通信形態で共通して使用することができることを示した。

SOS API の実装例として「**KOSEN 版ウェザーステーション**」への実装を行なった。本装置の通信形態は本研究で検討された通信形態と類似しておりそれを適用することで容易に実装する事が出来た。新規に開発するフィールドセンサへの SOS API 実装だけでなく、既存のフィールドセンサも工夫することで SOS API に対応できることを示した。

本研究で開発したソフトウェア群は図 22 に示すように GitHub 上で公開している［13］。SOS API の実装が容易になることで通信とデータの標準化が加速し、フィールドセンサの低価格化および充実した観測体制の構築につながることを願う。

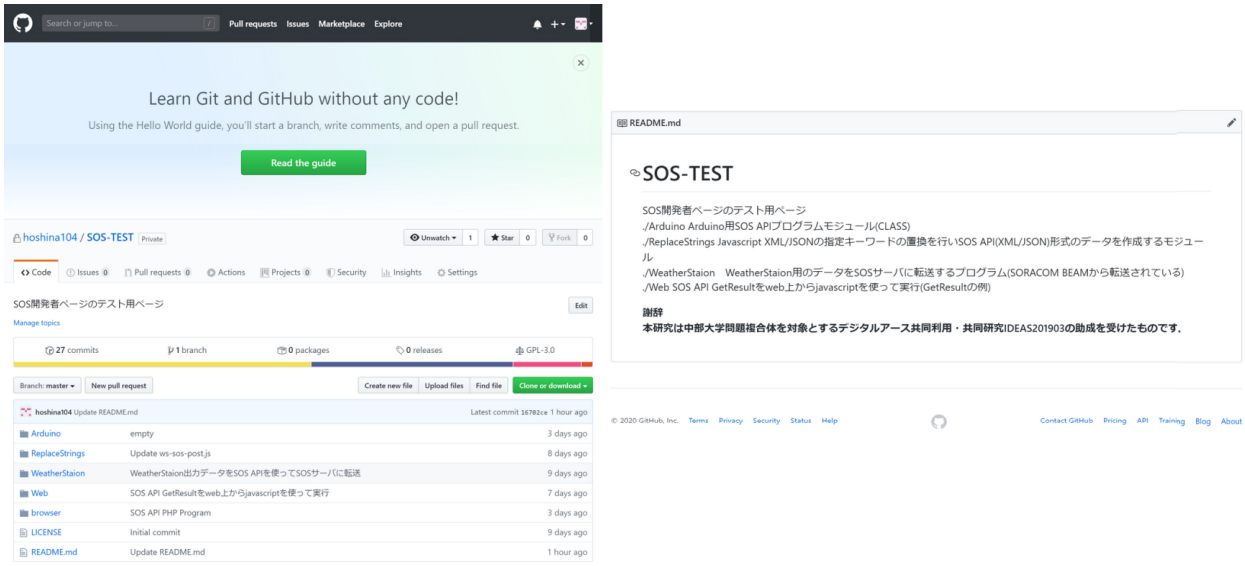

図 22 GitHub 上でのプログラム公開

6. 謝辞

本研究は中部大学問題複合体を対象とするデジタルアース共同利用・共同研究 IDEAS201903 の助成を受けたものです。

参考文献・データ

[1] “Open Geospatial Consortium,”: https://www.ogc.org/ (2020 年 4 月 1 日)

[2] “Sensor Observation Service,”: https://www.ogc.org/standards/sos (2020 年 4 月 1 日)

[3] C. S. J. E. Arne Bröring, “OGC® Sensor Observation Service Interface Standard,”: https://www.ogc.org/standards/sos (2020 年 4 月 1 日)

[4] “ICT でつなげる地域共生アグリ・バリュースペース研究開発プラットフォーム,”: https://www.agrivaluespace.com/core-ws/ (2020 年 4 月 1 日)

[5] “ｅコミュニティ・プラットフォーム,”: https://ecom-plat.jp/ (2020 年 4 月 1 日)

[6] 防災科学技術研究所, “相互運用ｇサーバー基本設計書,” 1 10 2015.: https://ecom-plat.jp/fbox.php?eid=17148&s=o (2020 年 4 月 1 日)

[7] “高度情報通信ネットワーク社会推進戦略本部（ＩＴ総合戦略本部） 農業分野における成果物等,”: https://www.kantei.go.jp/jp/singi/it2/senmon_bunka/nougyou.html (2020 年 4 月 1 日)

[8] “52°North Sensor Observation Service (SOS),”: https://github.com/52North/SOS (2020 年 4 月 1 日)

[9] “OSGeoLive,”: https://live.osgeo.org/en/index.html (2020 年 4 月 1 日)

[10] “npm express,”: https://www.npmjs.com/package/express (2020 年 4 月 1 日)

[11] “秋月電子通商,”: http://akizukidenshi.com/catalog/g/gK-09758/ (2020 年 4 月 1 日)

[12] “jQuery,”: https://jquery.com/ (2020 年 4 月 1 日)

[13] 保科紳一郎, “SOS-TEST SOS 開発者ページのテスト用ページ,”: https://github.com/hoshina104/SOS-TEST (2020 年 4 月 1 日)